Концептуальные элементы объектно-ориентированного программирования

Объектно-ориентированная технология основывается на так называемой *объектной модели.* Основными ее принципами являются: абстрагирование, инкапсуляция, модульность, иерархичность, типизация, параллелизм и сохраняемость. Каждый из этих принципов сам по себе не нов, но в объектной модели они впервые применены в совокупности.

Объектно-ориентированный анализ и проектирование принципиально отличаются от традиционных подходов структурного проектирования: здесь нужно по-другому представлять себе процесс декомпозиции, а архитектура получающегося программного продукта в значительной степени выходит за рамки представлений, традиционных для структурного программирования. Отличия обусловлены тем, что структурное проектирование основано на структурном программировании, тогда как в основе объектно-ориентированного проектирования лежит методология объектно-ориентированного программирования

Методы структурного проектирования помогают упростить процесс разработки сложных систем за счет использования алгоритмов как готовых строительных блоков. Аналогично, методы объектно-ориентированного проектирования созданы, чтобы помочь разработчикам применять мощные выразительные средства объектного и объектно-ориентированного программирования, использующего в качестве блоков классы и объекты.

Объектно-ориентированное программирование - это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

В данном определении можно выделить три части:

1) ООР использует в качестве базовых элементов объекты, а не алгоритмы (иерархия быть частью);

2) каждый объект является экземпляром какого-либо определенного класса;

3) классы организованы иерархически. Программа будет объектно-ориентированной только при соблюдении всех трех указанных требований. В частности, программирование, не основанное на иерархических отношениях, не относится к ООР, а называется программированием на *основе абстрактных типов данных*.

Для объектно-ориентированного стиля концептуальная база – *это объектная модель*. Она имеет четыре главных элемента:

- абстрагирование;
- инкапсуляция;
- модульность;
- иерархия.

Эти элементы являются главными в том смысле, что без любого из них модель не будет *объектно-ориентированной*. Кроме главных, имеются еще три дополнительных элемента:

- типизация;
- параллелизм;
- сохраняемость.

Называя их *дополнительными,* имеем в виду, что они полезны в объектной модели, но не обязательны.

Абстрагирование является одним из основных методов, используемых для решения сложных задач. Хоар считает, что "абстрагирование проявляется в нахождении сходств между определенными объектами, ситуациями или процессами реального мира, и в принятии решений на основе этих сходств, отвлекаясь на время от имеющихся различий"

Абстракция выделяет существенные характеристики некоторого объекта, отличающие его от всех других видов объектов и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя.

По мнению Сейдвица и Старка "существует целый спектр абстракций, начиная с объектов, которые почти точно соответствуют реалиям предметной области, и кончая объектами, не имеющими право на существование". Вот эти абстракции, начиная от наиболее полезных к наименее полезным:

· Абстракция сущности	Объект представляет собой полезную модель некой сущности в предметной области
· Абстракция поведения	Объект состоит из обобщенного множества операций
· Абстракция виртуальной машины	Объект группирует операции, которые либо вместе используются более высоким уровнем управления, либо сами используют некоторый набор операций более низкого уровня
· Произвольная абстракция	Объект включает в себя набор операций, не имеющих друг с другом ничего общего

Инкапсуляция

Абстракция и инкапсуляция дополняют друг друга: абстрагирование направлено на наблюдаемое поведение объекта, а инкапсуляция занимается внутренним устройством. Чаще всего инкапсуляция выполняется посредством скрытия информации, то есть маскировкой всех внутренних деталей, не влияющих на внешнее поведение. Обычно скрываются и внутренняя структура объекта, и реализация его методов.

Инкапсуляция, таким образом, определяет четкие границы между различными абстракциями.

Понятие модульности. По мнению Майерса "Разделение программы на модули до некоторой степени позволяет уменьшить ее сложность... Однако гораздо важнее тот факт, что внутри модульной программы создаются множества хорошо определенных и документированных интерфейсов. Эти интерфейсы неоценимы для исчерпывающего понимания программы в целом"

Программист должен находить баланс между двумя противоположными тенденциями: стремлением скрыть информацию и необходимостью обеспечения видимости тех или иных абстракций в нескольких модулях. Парнас, Клеменс и Вейс предложили следующее правило: "Особенности системы, подверженные изменениям, следует скрывать в отдельных модулях; в качестве межмодульных можно использовать только те элементы, вероятность изменения которых мала. Все структуры данных должны быть обособлены в модуле; доступ к ним будет возможен для всех процедур этого модуля и закрыт для всех других. Доступ к данным из модуля должен осуществляться только через процедуры данного модуля". Другими словами, следует стремиться построить модули так, чтобы объединить логически связанные абстракции и минимизировать взаимные связи между модулями. Исходя из этого, приведем определение модульности:

Модульность - это свойство системы, которая была разложена на внутренне связные, но слабо связанные между собой модули.

Таким образом, принципы абстрагирования, инкапсуляции и модульности являются взаимодополняющими. Объект логически определяет границы определенной абстракции, а инкапсуляция и модульность делают их физически незыблемыми.

Иерархия

Абстракция - вещь полезная, но всегда, кроме самых простых ситуаций, число абстракций в системе намного превышает наши умственные возможности. Инкапсуляция позволяет в какой-то степени устранить это препятствие, убрав из поля зрения внутреннее содержание абстракций. Модульность также упрощает задачу, объединяя логически связанные абстракции в группы. Но этого оказывается недостаточно.

Значительное упрощение в понимании сложных задач достигается за счет образования из абстракций иерархической структуры. Определим иерархию следующим образом:

Иерархия - это упорядочение абстракций, расположение их по уровням.

Основными видами иерархических структур применительно к сложным системам являются структура классов (иерархия "is-a") и структура объектов (иерархия "part of").

Типизация

Понятие типа взято из теории абстрактных типов данных. Дойч определяет тип, как "точную характеристику свойств, включая структуру и поведение, относящуюся к некоторой совокупности объектов". Для наших целей достаточно считать, что термины тип и класс взаимозаменяемы [Тип и класс не вполне одно и то же; в некоторых языках их различают. Например, ранние версии языка Trellis/Owl разрешали объекту иметь и класс, и тип. Даже в Smalltalk объекты классов SmallInteger, LargeNegativeInteger, LargePositiveInteger относятся к одному типу Integer, хотя и к разным классам . Большинству смертных различать типы и классы просто противно и бесполезно. Достаточно сказать, что класс реализует понятие типа]. Тем не менее, типы стоит обсудить отдельно, поскольку они выставляют смысл абстрагирования в совершенно другом свете. В частности, можно утверждать, что:

Типизация - это способ защититься от использования объектов одного класса вместо другого, или по крайней мере управлять таким использованием.

Типизация заставляет нас выражать наши абстракции так, чтобы язык программирования, используемый в реализации, поддерживал соблюдение принятых проектных решений. Вегнер замечает, что такой способ контроля существенен для программирования "в большом"

Идея согласования типов занимает в понятии типизации центральное место.

Параллелизм

Есть задачи, в которых автоматические системы должны обрабатывать много событий одновременно. В других случаях потребность в вычислительной мощности превышает ресурсы одного процессора. В каждой из таких ситуаций естественно использовать несколько компьютеров для решения задачи или задействовать многозадачность на многопроцессорном компьютере. Процесс (поток управления) - это фундаментальная единица действия в системе. Каждая программа имеет по крайней мере один поток управления, параллельная система имеет много таких потоков: век одних недолог, а другие живут в течении всего сеанса работы системы. Реальная параллельность достигается только на многопроцессорных системах, а системы с одним процессором имитируют параллельность за счет алгоритмов разделения времени.

Кроме этого "аппаратного" различия, мы будем различать "тяжелую" и "легкую" параллельность по потребности в ресурсах. "Тяжелые" процессы управляются операционной системой независимо от других, и под них выделяется отдельное защищенное адресное пространство. "Легкие" сосуществуют в одном адресном пространстве. "Тяжелые" процессы общаются друг с другом через операционную систему, что обычно медленно и накладно. Связь "легких" процессов осуществляется гораздо проще, часто они используют одни и те же данные.

Многие современные операционные системы предусматривают прямую поддержку параллелизма, и это обстоятельство очень благоприятно сказывается на возможности обеспечения параллелизма в объектно-ориентированных системах. Например, системы UNIX предусматривают системный вызов fork, который порождает новый процесс. Системы Windows NT и OS/2 - многопоточные; кроме того они обеспечивают программные интерфейсы для создания процессов и манипулирования с ними.

Лим и Джонсон отмечают, что "возможности проектирования параллельности в объектно-ориентированных языках не сильно отличаются от любых других, - на нижних уровнях абстракции параллелизм и ООР развиваются совершенно независимо. С ООР или без, все традиционные проблемы параллельного программирования сохраняются" . Действительно, создавать большие программы и так непросто, а если они еще и параллельные, то надо думать о возможном простое одного из потоков, неполучении данных, взаимной блокировке и т.д.

К счастью, как отмечают те же авторы далее: "на верхних уровнях ООР упрощает параллельное программирование для рядовых разработчиков, пряча его в повторно-используемые абстракции" . Блэк и др. сделали следующий вывод: "объектная модель хороша для распределенных систем, поскольку она неявно разбивает программу на распределенные единицы и сообщающиеся субъекты" .

В то время, как объектно-ориентированное программирование основано на абстракции, инкапсуляции и наследовании, параллелизм главное внимание уделяет абстрагированию и синхронизации процессов . Объект есть понятие, на котором эти две точки зрения сходятся: каждый объект (полученный из абстракции реального мира) может представлять собой отдельный поток управления (абстракцию процесса). Такой объект называется активным. Для систем, построенных на основе OOD, мир может быть представлен, как совокупность взаимодействующих объектов, часть из которых является активной и выступает в роли независимых вычислительных центров. На этой основе дадим следующее определение параллелизма:

Параллелизм - это свойство, отличающее активные объекты от пассивных.

Отношения двух любых объектов основываются на предположениях, которыми один обладает относительно другого: об операциях, которые можно выполнять, и об ожидаемом поведении. Особый интерес для объектно-ориентированного анализа и проектирования представляют два типа иерархических соотношений объектов:

- связи;

- агрегация.

Зайдевиц и Старк назвали эти два типа отношений отношениями старшинства и "родитель/потомок" соответственно .

Понятия класса и объекта настолько тесно связаны, что невозможно говорить об объекте безотносительно к его классу. Однако существует важное различие этих двух понятий. В то время как объект обозначает конкретную сущность, определенную во времени и в пространстве, класс определяет лишь абстракцию существенного в объекте.

В общепонятных терминах можно дать следующее определение класса: "группа, множество или вид с общими свойствами или общим свойством, разновидностями, отличиями по качеству, возможностями или условиями". В контексте объектно-ориентированного анализа дадим следующее определение класса:

Класс - это некое множество объектов, имеющих общую структуру и общее поведение.

Любой конкретный объект является просто экземпляром класса. Что же не является классом? Объект не является классом, хотя в дальнейшем увидим, что класс может быть объектом. Объекты, не связанные общностью структуры и поведения, нельзя объединить в класс, так как по определению они не связаны между собой ничем, кроме того, что все они объекты.

Большинство объектно-ориентированных языков непосредственно поддерживают разные комбинации следующих видов отношений в классах:

ассоциация
наследование
агрегация
использование
инстанцирование
метакласс.

Классы и объекты - это отдельные, но тесно связанные понятия. В частности, каждый объект является экземпляром какого-либо класса; класс может порождать любое число объектов. В большинстве практических случаев классы статичны, то есть все их особенности и содержание определены в процессе компиляции программы. Из этого следует, что любой созданный объект относится к строго фиксированному классу. Сами объекты, напротив, в процессе выполнения программы создаются и уничтожаются.

На этапе анализа и ранних стадиях проектирования решаются две основные задачи:

Выявление классов и объектов, составляющих словарь предметной области.

Построение структур, обеспечивающих взаимодействие объектов, при котором выполняются требования задачи.

В первом случае говорят о ключевых абстракциях задачи (совокупность классов объектов), во втором - о механизмах реализации (совокупность структур).

На ранних стадиях внимание проектировщика сосредоточивается на внешних проявлениях ключевых абстракций и механизмов. Такой подход создает логический каркас системы: структуры классов и объектов. На последующих фазах проекта, включая реализацию, внимание переключается на внутреннее поведение ключевых абстракций и механизмов, а также их физическое представление. Принимаемые в процессе проектирования решения задают архитектуру системы: и архитектуру процессов, и архитектуру модулей.